AFRL-HE-WP-TP-1998-0007

UNITED STATES AIR FORCE RESEARCH LABORATORY

Intelligent Agent Feasibility Study Volume 1: Agent-based System Technology

Stephen E. Deutsch

BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge MA 02138

February 1998

Final Report for the Period November 1997 to February 1998

Approved for public release; distribution is unlimited.

Human Effectiveness Directorate
Crew Survivability and Logistics Division
Sustainment Logistics Branch
2610 Seventh Street
Wright-Patterson AFB OH 45433-7901

NOTICES

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner, licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

Please do not request copies of this paper from the Air Force Research Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Federal Government agencies and their contractors registered with Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
8725 John J. Kingman Rd., STE 0944
Ft Belvoir VA 22060-6218

DISCLAIMER

This Technical Paper is published as received and has not been edited by the Air Force Research Laboratory, Human Effectiveness Directorate.

TECHNICAL REVIEW AND APPROVAL

AFRL-HE-WP-TP-1998-0007

This paper has been reviewed by the Office of Public Affairs (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical paper has been reviewed and is approved for publication.

FOR THE COMMANDER

For Lt Col
THOMAS J. MOORE, Chief
Crew Survivability and Logistics Division
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1998	Final - November 1997 - February 1998

4. TITLE AND SUBTITLE
Intelligent Agent Feasibility Study Volume 1; Agent-based System Technology

5. FUNDING NUMBERS
C - F41624-97-D-5002
PE - 62202F
PR - 1710
TA - D0
WU - 04

6. AUTHOR(S)
Stephen E. Deutsch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER
BBN No. 8216

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory, Human Effectiveness Directorate
Crew Survivability and Logistics Division
Air Force Materiel Command
Sustainment Logistics Branch
Wright-Patterson AFB OH 45433-7604

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-HE-WP-TP-1998-0007

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Research on software agents and agent-based systms has now been underway for more than twenty years. Within the last few years interest in agent-based systems has spread to include the active involvement of the broader commercial software community. In Volume 1 of this study we have traced these developments in the academic, commercial, and government arenas. To provide the necessary background for the study we have examined the definition of agents and the support for agent-based systems in client-server and distributed object computing environments. The focus of the study was the investigation of potential applications for agent-based technologies within the logistics areas of deployment and maintenance. An agent-based architecture was developed to support the application areas. The architecture draws on the services of the newly emerging distributed object computing environments and the combined proactive, reactive and multi-tasking agent capabilities of the Operator Model Architecture (OMAR) agent. Volume 2 provides an examination of the Mission Capable Awaiting Parts (MICAP) maintenance process as it is conducted today and outlines recommendations for employing agent-based system capabilities to improve the process.

14. SUBJECT TERMS
intelligent agents; agent-based architectures; human computer interfaces; agents; operator model architectures; OMAR

15. NUMBER OF PAGES
48

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94

Preface

The work on the Intelligent Agent Feasibility Study was conducted under Delivery Order 2 of the Logistics Technology Research Support (LTRS) program administered under U. S. Air Force Contract Number F41624-97-D-5002.

The author wishes to thank the Contract Monitor, Mr. Michael J. Young of Armstrong Laboratory's Human Resources Directorate, for his constructive criticism and enthusiastic support of the research effort.

Table of Contents

Figures

1. Introduction and Overview

The subject of this study, agents and agent-based systems, has been an active research area for many years, but it is only recently that this work has attracted significant attention. Agent-based systems are now under development as commercial products, software agents are being added to existing commercial products, and academic and commercial laboratories are expanding their research on agents with a particular emphasis on Internet applications. Our purpose in this study has been to examine potential uses for agents and agent-based systems in Air Force logistics operations. Durfee (1991), as editor, provides an informative retrospective on important early research efforts in the area of distributed artificial intelligence with several papers (Zlotkin & Rosenschein, 1991; Lesser, 1991; Pan & Tenenbaum, 1991; Hewitt & Inman, 1991) discussing distributed agent operation.

We start, in Section 2, by identifying the important attributes that define agents and differentiate them from objects in object-oriented systems. As we will see, many of the factors cited in attempting to distinguish agents from objects are attributes that might well be claimed for objects as well. Agents may well be viewed as specialized objects with the potential to operate seamlessly in an object-oriented environment. The close relationship between agent-based systems and object-oriented systems is important. A second important relationship is highlighted by Lesser's (1991) focus on distributed problem solving. In a more modern guise, distributed problem solving will be accomplished in a distributed object computing environment. The development of agent-based systems, with agents designed as extensions of objects, will find significant support in the new work in distributed object systems. These important links between agent-based systems, object-oriented systems and distributed object systems are outlined in Section 2.

Section 2 will also provide a look at a selected group of agent-based systems under development. Of the three systems from commercial companies, two are from research laboratories and are unsupported at this time, while one has recently become available as a supported product. While much of the effort in the development of these systems has been devoted to supporting the mobility of their agents, the agents themselves can make only modest claims to important agent attributes beyond this mobility. In several agent-based systems, much of their capability is provided by the language that is the basis for communication among their agents. To explore this aspect of agent-based systems we have looked at the Agent Communication Language, the Interagent Communication Language and the agent-based systems that are based on them. All of these systems are interesting in that they rely on facilitators that take on the task of matching an agent's service requests to agent service providers leading to greater system flexibility.

In Section 3 we look at potential Air Force logistics operations in which agent-based systems might be expected to play a role. Three potential application areas were investigated. The first involved virtual manufacturing systems under development to support the production of the Joint Strike Fighter and the C-17. This work was found to focus primarily on aircraft structural members. It does not yet address aircraft avionics and weapons systems. When virtual manufacturing addresses these systems it can be

AS AMENDED

expected to provide important support for maintenance procedure development and evaluation as part of the aircraft design process, and agent-based systems can be expected to play a role in that development. An evaluation of aircraft maintenance related to Mission Capable Awaiting Parts (MICAP) processing was the second application investigated. The third potential application was the Air Mobility Command's (AMC) Corporate Data Base initiative and the functional capabilities whose implementation is to be based on that initiative. The logistics maintenance and deployment arenas were each found to be staffed by duty officers that must address several concurrent tasks in situations where they are required to collaborate with superior officers, peers and subordinates, some of whom are at the local site while others are working at remote sites. Today, they are poorly supported by loosely coupled legacy systems. An important finding of this study is that agent-based systems operating within distributed object computing environments such as the AMC's Corporate Data Base can play an important role in the design and development of the collaborative systems that would support these critical activities.

Global Communications Strategy that initiates an AMC

Section 4 provides an outline of an architecture for an agent-based system using the Operator Model Architecture (OMAR) as a starting point and examining the architectural extensions required for operation in a distributed object computing environment. We begin by outlining the principal attributes of the proposed application environments that can be expected to impact the architecture for Distributed OMAR. We then address the site services typically provided for agent operation at the nodes in a network and the support services typically provided for distributed objects. The point is made that there may well be more than one agent-based system operating in a large distributed application environment. It will be important to be able to draw on the resources provided by the non-OMAR agents and, at the same time, provide OMAR services to these non-OMAR agents. Having discussed the system level consideration for the Distributed OMAR architecture, we next look at the OMAR agents themselves. Developed, as they were, for building models of human performance, they are capable of performing a mix of proactive and reactive behaviors and are capable of multi-tasking. These are viewed as essential attributes necessary to support the collaboration of human users in managing complex evolving situations.

Finally, in Section 5 we look at the issues of costs, risks and benefits. When examining intelligent agents it is easy to form expectations of new, innovative and "intelligent" solutions to difficult problems. Here, we have argued strongly that the benefits of agents and agent-based systems lie rather in their ability to provide a framework in which to design systems capable of interacting with users at remote sites to support the collaborative efforts of the users to accomplish their tasks. Software agents are resources that may provide information to support decision making, planning, task execution, and monitoring task progress and outcomes. Agents' strengths lie in their communication abilities among themselves and between themselves and their human users. These communication skills are the essential elements for good system design in distributed object systems. The multi-tasking capabilities of OMAR agents are essential for supporting the monitoring and management of the multiple threads of complex operations. As distributed object systems begin to come on-line, their operating

environments will provide many of the important services necessary to support the development of these agent-based capabilities. These factors combine to reduce costs and risks in implementing agents as distributed system components.

2. Agents and Agent-based Systems

Agents and agent-based systems are very closely related to objects and object-oriented programming environments. They are, in fact, so closely related that it has taken some effort on the part of researchers to satisfactorily differentiate them. This section begins by looking at the important attributes that define an agent and differentiate it from an object. There are also close ties between agent-based systems and the newly developed distributed object computing environments. Several of the services being developed for the various implementations of distributed object systems are essential to agent-based systems as well. This section continues by examining the relationship between agent-based systems and distributed object computing environments. The remainder of the section is devoted to examining several agent-based systems that are under development in commercial and academic research communities.

2.1 Agent Attributes

Intelligent agents, software agents, or just agents: however they are designated, they immediately provoke challenging questions. What are agents? How are they different from programs? Or, asked in a more modern version: How are they different from objects? Or, how are they different from objects in a distributed object computing environment? Franklin and Graesser (1996) provide a good survey of the attempts by the developers of a number of agent-based systems to address these questions. One of the more complete definitions that they cite is that of Wooldridge and Jennings (1995). Jennings and Wooldridge (1996) provided a slightly edited version of their "key hallmarks of agenthood:

- *Autonomy:* agents should be able to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they should have a degree of control over their own actions and their own internal state.
- *Social ability:* agents should be able to interact, when they deem appropriate, with other software agents and humans in order to complete their own problem solving and to help others with their activities where appropriate.
- *Responsiveness:* agents should perceive their environment (which may be the physical world, a user, a collection of agents, the Internet, etc.) and respond in a timely fashion to changes which occur in it.
- *Proactiveness:* agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behaviors and take the initiative where appropriate."

Jennings and Wooldridge (1996) then attempt to differentiate software agents from object-oriented systems, artificial intelligence and distributed computing by citing the high-level tasks that are delegated to agents and the degree of autonomy that the agents are granted in carrying out their tasks. They also point to the dynamically changing

environments in which agents operate. In the discussion that follows we take a viewpoint that downplays the distinction that Jennings and Wooldridge attempt to make. The development of agents owes much to earlier work in object-oriented programming and artificial intelligence, and agents will operate very naturally in distributed object system environments. The intelligence claimed for agents by referring to them as intelligent agents will always be open to question just as it has been in the field of artificial intelligence since its inception. Clearly, we would like the agents that we develop to be intelligent, and artificial intelligence research will provide a base from which to achieve some degree of intelligence in agent behaviors. Here, we will simply refer to agents as software agents or just agents, leaving aside the claim of intelligence for a later time.

Jennings and Wooldridge explicitly call attention to the requirement for communication not only among agents, but also communication between agents and humans. The applications that we will identify for agent-based systems will depend heavily on this important agent capability.

Green, Hurst, Nangle, Cunningham, Somers and Evans (1997) offer a more concise definition of agenthood: *"An agent is a computational entity which:*

- *acts on behalf of other entities in an autonomous fashion;*
- *performs its actions with some level of proactivity and/or reactiveness;*
- *exhibits some level of the key attributes of learning, co-operation and mobility."*

While addressing most of the attributes more fully stated by Jennings and Wooldridge, this definition adds learning and agent mobility as agent attributes. Learning is finding a place in agent applications, particularly in support of Web browsing. As progress is made in machine learning it can be expected to play a larger role in agent-based systems.

The argument for utility in agent mobility is based on sending an agent to a large data source, thereby eliminating extensive remote queries. Several issues complicate agent mobility. As we will see in looking at some of the early efforts in agent-based systems, some of the easier problems are being addressed. There are solutions to the problem of sending messages to agents that have moved on to another site, and self-initiated moving is available to simple agents. But it is easy to see that there may be cases in which an agent has a complex on-going effort at a local site and at the same time needs to extensively probe a large data base at a remote location. This is clearly an instance in which the remote data access might best be delegated to another agent, while current local activities are pursued. Agent mobility may turn out to be useful only in single-task agents. An agent seeking a resource or perhaps the best "price" for a resource that resides at one or more sites is an example.

The complex multi-tasking agent will more likely rely on communicating with remote agents to request extensive services at remote sites. The multi-tasking agent may have on-going activities at the local, site making it impractical to travel to a remote site to obtain information from a data base. The data retrieval can be delegated to a remote agent while the local agent carries on its concurrent tasks. A second complicating issue is the question of how the mobile agent's current state is to be reestablished at the new site. Agent mobility today is accomplished either by using simple scripting languages (Gray, 1996)

or by agents that reestablish operations based on minimal pre-move data. It is not yet clear just how one might reestablish the full computational state of a complex agent at a new site.

2.2 Distributed Object Systems

Agents and agent-based systems have been active research areas for many years. With parallel efforts in object-oriented and distributed computing now coming together in the form of distributed object computing environments, agent-based systems can now be viewed as a component technology in this broader field. Since distributed object computing environments will be mentioned frequently in subsequent sections, it is appropriate to pause here to provide a very brief outline of the key elements of distributed object systems.[1]

Diversity in hardware and software platforms has been a fact of life in computing systems from their earliest days. Until quite recently this heterogeneity has been an intractable problem. Two typically monolithic and geographically separated legacy systems, built on different platforms might well complement one another, but there was little chance of bringing them into a single system combining their features.

In stark contrast with yesterday's monolithic systems, more recently developed systems have been built from components based an object-oriented and client-server technology. Object-oriented and client-server technologies have also evolved to bring together legacy systems to access the functionality of each. Still newer systems, based in part on the merger of object-oriented and client-server technologies, are taking the form of distributed object computing environments. The developers of these computing environments are in the process of addressing a broad range of difficult problems.

In object-oriented systems, the objects have attributes or states and behaviors, with behaviors that are invoked by sending a message to the object. A typical object can respond to a number of messages—it can provide several behaviors or services. Related object types may share or inherit attributes and behaviors. A collection of objects may be developed and integrated to form a component with reusable functionality. A typical example in a word processing environment might be a spelling checker.

In client-server technology, a cluster of functionality is developed to form a server available at a node in a network. One or more users requiring that service, clients, may then access that service across the network. Remote access to a computing resource for the service is provided. Neither server nor client need be implemented using object-oriented technology. A server may be just a legacy system enhanced by a suitable wrapper that enables access to it in a network environment.

When a broad set of services is made available to maintain a network of heterogeneous network nodes such that they provide for the management and interaction of the objects

[1] The material on distributed object systems was developed from the following two Web pages developed by Alan L. Pope and Chuck Kollars respectively: http://www.qds.com/people/apope/ap_TheObject.html; http://www1.shore.net/~ckollars/distobjs.html.

active at those nodes, and multiple server-side and client-side capabilities are each developed as clusters of objects with behaviors operating in a networked environment, the composite is a distributed object computing environment. The network nodes may be heterogeneous in the sense that hardware and operating systems are different at different nodes, and the software languages for server and client components are also different. Graham (1993) provides a good overview of object technologies. Orfali, Harkey and Edwards (1996) provide a useful introduction to distributed object computing systems.

2.3 Agent-based Systems: Commercial Efforts

Several commercial companies are at work on agent-based systems. A group at an IBM research laboratory in Japan has developed Aglets, General Magic is at work on Odyssey and Object Space is working on Voyager. In concept the systems are very much alike. Each provides an agent-support environment at each site at which agents are to operate. Typically, the environment provides basic services for creating and destroying agents, passing messages between agents, and handling the movement of agents from one site to another. Hence, an agent that must conduct a high bandwidth exchange with a remote object may move to the site of the remote object so that the exchange becomes a local one with the attendant resource savings. More generally, an agent can move from site to site to obtain the resources it requires.

Work on agent-based systems is closely related to work on distributed object computing environments, the computing environment emerging as the successor to client-server systems. One would reasonably expect that as agent-based systems become available, their agents would be developed as extensions of an object in a distributed object system. Most noticeably this is not the case for passing messages between Aglets. In the Aglet system, an Aglet receiving a "message" must dispatch the message itself. That is, user-written code must examine the message, decide whether or not it can process the message and then select the routine to act on the message, a task typically managed by the messaging system itself. This is a somewhat surprising retreat from an essential object-based protocol in this IBM research effort.

In the context of an intelligent agent study, it is important to examine the contribution to the technology made by these emerging systems. Clearly their focus is on providing a software framework in which an agent can operate, and a software environment that is supported at each site at which agents are to be created and operate. In the Voyager system, agents can be created at remote sites. In the Aglets and Odyssey systems an agent can only be created locally and then must be moved to the desired remote site. As outlined below, Voyager provides "secretaries" to facilitate the tracking of agents as they move from site to site, but the other systems require that developers generate the code to keep track of moving agents. Typically, each agent operates in its own thread and it is this thread that constitutes the capabilities of the agent. Beyond providing the basic services outlined here, these systems provide no particular services to support the development of agent capabilities.

A review of these agent-based systems is valuable in that it provides insight into the development of these essential agent services. Unfortunately, what we have seen is that

what the agents actually do is left entirely to the application developer. Indeed, one of these initial agent-based systems, IBM's Aglets, has a made significant retreat from what are today standard message passing protocols. As we have seen, an agent may be created to operate locally or to move to another site, but at that point it is up to the developer to produce the code to accomplish the agent's objectives. This conception of an agent has very little to do with "intelligence"; it is simply about managing the execution of a code thread at a site, perhaps enabling the code thread or some "reduced" version of it to be moved to another supported site where it is to resume execution. These systems are valuable to the extent that they provide a road map for the establishment of these basic services for the creation and support for agents in a distributed object computing environment. The following sections provide a brief overview of three of these agent-based systems: IBM's Aglets, General Magic's Odyssey and Object Space's Voyager.

2.3.1 IBM's Aglets

IBM's Aglets[2] is a Java-based system in which the agents, or Aglets, are program code and data that can execute at a network site, cease execution, move from that site in the network to another, and resume execution. An itinerary object is used to manage the movement of the Aglet within the network. Messages that are objects themselves can be sent to Aglets to generate reactive behaviors. However, it appears that the "messages" are not messages in the traditional object-oriented or distributed object sense. An Aglet message must be assessed by the Aglet to determine if it can be handled and then dispatched and operated on if appropriate. An Aglet responds with true or false to indicate whether or not it has handled the message and then uses an explicit method to provide the response. Message passing for Aglets may take several forms. For synchronous messages, Aglets wait for a response, while the response to asynchronous messages may be retrieved at a later time, allowing the sender to proceed with its execution. While advertised as asynchronous message response handling, the developer must put in place the code to continue code execution while polling for the message response.

There are a number of basic operations that can be performed on Aglets: they can be created, dispatched to remote network locations, retracted from their remote location, deactivated and reactivated, and finally disposed of. Retracting an Aglet from a remote site is an asynchronous operation to retrieve an Aglet on demand rather than waiting for it to return on its own initiative. The deactivation of an Aglet refers to the temporary storage of the Aglet on a secondary storage device. For each of these Aglet operations provisions are made to execute specialized "housekeeping" code before and/or after the operation as necessary. An Aglet can also be cloned to produce a new Aglet that differs in name.

Aglet operations take place within the context of an agent support system. New Aglets or clones are created only within the local context. For each Aglet within a context, there is a proxy for the Aglet, and it is the proxy through which Aglet operations are invoked

[2] The material on IBM Aglets was developed from the Web pages at http://www.trl.ibm.co.jp/aglets/.

rather than on the Aglet itself. The operations on Aglets outlined above are invoked on the proxy for the selected Aglet. Contact with Aglets at remote sites is also handled through the proxies for those Aglets. When Aglets are dispatched to remote sites or move on their own, they must be tracked or searched for by those Aglets wishing to make further message-passing contact with them.

Prioritization and parallel execution for Aglets is handled at the message level. An Aglet can assign a priority to a message type. This is somewhat limited by the fact that the message type is simply specified as a string on the type slot variable of the message. It does not appear to be possible to group messages of a given type as a subclass of the generic message object. Having assigned priorities to particular message types, messages will be delivered to the Aglet in the order determined by message priority. Parallelism is provided by allowing a given code block processing a message to release the message handler to start the processing of another message by the Aglet in a separate thread.

2.3.2 General Magic's Odyssey

Odyssey is a Java-based agent system developed by General Magic[3]. An agent is loosely defined as having its own thread of execution and the ability to "act autonomously on behalf of a person or organization." In Odyssey, an agent is created in and operates in an environment referred to as a *place*. Agents are mobile and may move from one place to another. A set of agents and places make up an agent system. An Odyssey *worker* class, a subclass of the agent class, is said to simplify the construction of mobile agents. A worker has an itinerary with a task to be performed at each stop on the itinerary. Each worker can manipulate its own itinerary to dynamically alter its list of destinations or the tasks to be executed at each destination.

Odyssey uses the Remote Machine Interface (RMI) as its default transport mechanism. It also supports Microsoft's Distributed Component Object Model (DCOM) or the Object Management Group's (OMG) Internet Inter-Orb Protocol (IIOP).

2.3.3 Object Space's Voyager

Voyager is a distributed object system developed by Object Space[4] that supports agent-based computing. Written in Java, Voyager's agents are Java objects. In distributed object fashion, agents can be created either locally or remotely and messages may be sent to either local or remote agents. Synchronous messages are messages in the traditional sense in which the sender resumes execution when the response to the message arrives. Voyager also supports *oneway* and *future* messages. When issuing a *oneway* message the sender resumes immediately. When using a *future* message the sender resumes immediately, having set up a place holder that allows the result to be retrieved at a later point in time.

---

[3] The material on General Magic's Odyssey System was developed from the Web pages at http://www.genmagic.com/.

[4] The material on Object Space's Voyager System was developed from the Web pages at http://www.objectspace.com/voyager/.

Within Voyager, agents are distinguished from the objects from which they are constructed by the ability to move themselves within the network. The agent's method to be executed upon arrival at the new site is be specified before departure. Voyager also addresses the problem of tracking mobile agents. Mobile agents leave behind a *secretary* at the site from which they depart. The agent's secretary takes on the responsibility of forwarding messages addressed to its agent and provides the new site location of its agent to the senders of the messages that the secretary has forwarded.

Object Space is also at work on an integration with the Common Object Request Broker Architecture (CORBA). They are planning to maintain Voyager's mobile agent programming capabilities while at the same time providing access to CORBA objects. In like manner, Object Space is working to make Voyager objects available for remote access by CORBA objects. Object Space's goal is to make the CORBA-ness of these references transparent to Voyager code developers.

Given this short synopsis, it can be seen that in Voyager, Object Space has thoughtfully provided an agent capability as an extension of a distributed computing environment. The mechanics of creating agents, providing them with access to distributed objects in the network, enabling them to reference other agents in the same manner as basic objects are referenced, and enabling them to move within the network appear to be well worked out. Voyager provides a good model of how agent capabilities can be provided in a distributed computing environment.

2.4 The Agent Communication Language

Agents and agent-based software have been defined in a number of different ways (Franklin & Graesser, 1996). Genesereth and Ketchpel (1994) use the communication language for agents as the basis of their definition of software agents. Agents are just those "software components that communicate with their peers by exchanging messages in an expressive agent *communication language*." Software development is then the creation of application programs written as software agents. The agents developed typically operate in separate threads or processes spread across one or more machines.

Genesereth and Ketchpel highlight the distinction between software agents and object-oriented programming. Both are based on message passing and both use messages to eliminate the need for a consumer to understand the internal structure of the provider. However, unlike the typical object-oriented programming environment in which each object is free to interpret the meaning of a message, the messages of agent-based systems use the shared semantics of a common language that is agent-independent.

The particular communication language discussed by Genesereth and Ketchpel is the Agent Communication Language (ACL) developed as part of the DARPA Knowledge Sharing Effort (Neches, Fikes, Finin, Gruber, Patil, Senator, & Swartout, 1991). ACL is assembled from three constituents: a vocabulary of defined dictionary terms; an inner language, the Knowledge Interchange Format (KIF); and an outer language, the Knowledge Query and Manipulation Language (KQML). An ACL message is then "a KQML expression in which the 'arguments' are terms or sentences in KIF formed from words in the ACL vocabulary" (Genesereth and Ketchpel, 1994). The words of the

vocabulary each have two definitions: the first, an English description specifying the meaning of the word for use by humans; and the second, a formal annotation in KIF for use by programs.

The agent-based software described by Genesereth and Ketchpel was developed in part to address interoperability problems that frequently plague large systems that have grown over a long period time, where system components, written in different languages, have been added or deleted as system requirements evolved. Here, there is a need for software agents to interact with legacy software. Three approaches to constructing the interface to the legacy systems have been considered by Genesereth and Ketchpel (1994). In the first, a *transducer* is developed as the interface between an agent and a component of legacy code. Based on a message from another agent, the transducer converts the message into a call acceptable to the legacy system, and handles the response from the legacy system, making the necessary adjustments to respond to the requesting agent. This approach has been used effectively to build graphical user interfaces to legacy systems. The second approach uses a *wrapper* to enable the legacy system to communicate in ACL. In building the wrapper, one takes on the burden of working with the program and data structure of the legacy program, but gains some efficiency in system operation over the transducer approach. The third approach is not an interface at all, but rather the rewriting of the legacy system component as agent-based software.

When considering the organization of the agents or architectures for agent-based systems, Genesereth (1992) advocates the use of *facilitators*. Rather than allowing agents to communicate directly with other agents, agents communicate with one another through the facilitators, where each facilitator may serve a group of one or more agents. Facilitators provide meta-level and application-level capabilities. At the meta level, agents provide their facilitators with information describing their capabilities and requirements. At the application level, agents make requests and respond to requests through their facilitators. Communication between agents and facilitators is conducted using ACL. It is the task of the facilitators, using the agents' meta-level information, to route requests to agents providing appropriate services. The use of facilitators represents a significant extension to the site services typically provided by agent-based systems. The feature that distinguishes facilitators is their capability to use the meta-level information from agents as the basis for routing message-based agent requests for services. In the typical agent-based or distributed object system a consumer must "know" the name of the service provider and the message to transmit to obtain the desired service. Genesereth describes the facilitator-based architecture as a *federated system*. The term federated system is frequently used more loosely to refer to an assembly of legacy systems and data base resources typically integrated and managed using an object-oriented superstructure with a user interface providing access to the legacy functionality.

2.5 The Open Agent Architecture

The Open Agent Architecture (OAA) (Cohen, Cheyer, & Wang, 1994), under development at SRI International, is an agent-based system closely related to the work based on ACL, but more ambitious in its scope. In OAA (Moran, Cheyer, Julia, Martin,

& Park, 1997), agents can operate on multiple platforms across a network, new agents can be added to an operating environment, and agents can be removed, either by design or through partial network or system failures. Support is available so that OAA agents may be developed in a broad range of languages: C, Prolog, Java, Visual Basic, and Borland's Delphi. Communication between agents uses the SRI-developed Interagent Communication Language (ICL), a language based on Prolog. ICL extensions to Prolog are used to address temporal information. The user interface to OAA, itself developed as OAA agents, can run on a PDA facilitating access to OAA services from remote sites. User input modalities extend beyond the usual keyboard and mouse input to include speech, handwriting and pen-based gestures. Interpretation of these inputs is handled by additional OAA agents.

OAA uses a blackboard framework, implemented as a server, with the OAA agents as the clients. The server maintains data that is global to the agents. The server process is responsible for "identifying agents that can achieve various goals, and for scheduling and maintaining the flow of communication during distributed computation" (Cohen et al., 1994). As in the ACL-based system, many of these tasks are handled by *facilitator* agents (Moran et al., 1997). As new agents are added to the system they are required to register their capabilities with a facilitator. Part of this process includes the registration of the natural language vocabulary used to specify the agent's capabilities. It is then the responsibility of the facilitator to route consumer-agent service requests to appropriate agent providers. The developers envision communities of agents in which atomic requests from one agent community may be decomposed to be addressed by several agents in another community. The decomposition is the responsibility of a facilitator operating on ICL requests using declared agent capabilities. Importantly, agents can be responsive to *triggers*, taking appropriate action in response to a specified event.

2.6 Agent Coordination and Negotiation

While interest in multi-agent systems is a recent phenomenon, the areas of agent coordination and agent negotiation have been active research areas for quite a few years. One reason for the research interest in multi-agent systems stems from their potential to provide solutions to complex problems through distributed problem solving. Here, agent coordination plays an important role and can lead to the requirement for agent negotiation. Various approaches to agent coordination have been examined. The most traditional are hierarchical, based on a master agent directing the activities of one or more slaves. In more decentralized approaches, as discussed by Green et al. (1997), agents can take on multiple roles as in the Contract-Net Protocol (CNP) defined by Smith and Davis (1981; Davis & Smith, 1983). Following the CNP protocol, if a problem can't be solved locally, help is sought through a contracting mechanism by which the managing agent announces the requirements for services and contracting agents bid to take on the task. While the interaction between agents as originally defined in CNP consists of a selection based on a single submission of bids, it is easy to see that the selection process could evolve to include adjustments to the original requirements and the submission of several rounds of bids, that is, negotiation.

Green et al. (1997) provide an overview of multi-agent systems composed of cooperative as well as self-interested agents. They also provide additional information and references on agent coordination and agent negotiation. Coordination, while still an ongoing research area, is operationalized to form an essential element in all agent-based systems. It is complicated by the move to more decentralized systems and further complicated by the certain emergence of systems that are heterogeneous in the sense that they include agents from more than one agent-based system. As federated systems evolve, bringing together legacy systems, it should not be surprising to find that services provided by individual legacy systems are provided through agent-based systems. As a result, a large federated system may include more than one agent-based system.

As with agent coordination, research in agent negotiation has yet to mature or be fully explored. The delegation of the authority to negotiate on behalf of the user will be a sensitive issue for some time. The users of newly emerging systems may be ready to have agents assemble data and make recommendations for their review, but they may be justified in their reluctance to delegate responsibility for negotiation in any but the most trivial interchanges. On the other hand, an agent-based system designed to meet anything more than the most immediate requirements of a particular application should provide an environment capable of supporting research in agent coordination and negotiation.

AS AMENDED

3. Agent-based System Application Domains

The investigation of potential application domains in which agents or agent-based systems might play a role has been an important part of this study. The three application areas investigated included virtual manufacturing, the Air Mobility Command's Corporate Data Base effort, and support for the management of Mission Capable Awaiting Parts (MICAP) operations in aircraft maintenance.

Global Communications Strategy

3.1 Virtual Manufacturing

3.1.1 Relating Virtual Manufacturing to Logistics and Agent-based Systems

Virtual manufacturing is about building a link between the design process for a product and the manufacturing process for that product. An important part of that link is the use of design data as input to the simulation of the manufacturing process. There are a number of important gains that will be made possible through this integration, one of which has significant bearing on logistics. The area that we are concerned with is the interaction between the design process and the manufacturing process. By working on these product tasks in parallel, manufacturing concerns can be addressed at a time when they can have a positive impact on the design process: design changes can be made that will improve the speed and reduce the costs of the manufacturing process. It is exactly this potential that we would like to realize in the maintenance area. The ability to author maintenance procedures in parallel with the product design process would make it possible to realize similar savings in the logistics arena.

Important aspects of the manufacturing process are directly related to the process of developing maintenance procedures. Design for assembly is probably the best

representative example. There is a clear link here between a manufacturing process and a maintenance process. The computer-based tools that support one can be adapted to support the other. Product assembly and by extension product disassembly are critical maintenance tasks. Virtual manufacturing software tools designed to support the verification of assembly/disassembly can immediately support maintenance procedure development. In looking at virtual manufacturing it is this synergy that we hope to exploit.

The simulation of the manufacturing process based on product data will make significant demands on the form that the product data takes. If programs are to contribute to, for example, the evaluation of the assembly process, they will require a semantically labeled product model. In developing the disassembly process to access a replaceable part, the CAD constructs that define the panel covering the part will have to be identifiable as such, as will the fasteners holding the panel in place. Upon the removal of the panel, additional components preventing access to the part of interest will also require appropriate labeling. These are some of the requirements that will be addressed in developing the link between design and manufacturing, and these are the ones that might be exploited to support maintenance procedure development.

3.1.2 Virtual Manufacturing Efforts Examined

3.1.2.1 The McDonnell Douglas CSTAR Program

The McDonnell Douglas C-17 STEP Transfer and Retrieval (CSTAR[5]) and CSTAR2[6] programs have the goal of providing electronic data exchange between C-17 facilities in Long Beach, California and St. Louis, Missouri. The data are being exchanged using STEP (Standard for Exchange of Product model data—ISO 10303). The STEP application protocol AP203 that provides configuration-controlled 3D design of mechanical parts and assemblies is being used. CORBA provides the network service layer linking remote locations. Previously employed manual data transfer methods required a staff of three people, were error prone and typically took three weeks to two months to complete. Data delivery is now accomplished overnight without people dedicated to support the process. The initial data transfers also included data on the inboard and outboard pylons for the C-17. The data transfer included configuration management information. More recent work has included the transfer of data related to the C-17 troop door air deflectors, the main landing gear pod and bulkheads, the cargo ramp torque box, and canted bulkheads. The design data being handled in the CSTAR program doesn't yet include the avionics or weapons system equipment that would be more closely related to maintenance logistics operation support. CSTAR does not include any agent-based software components.

---

[5] The material on the CSTAR program was developed from the Web pages at http://www.scra.org/pdesinc/events/c17.html and http://www.scra.org/pdesinc/annreprt.html.

[6] The material on the CSTAR2 program was developed from the Web page at http://bikini.scra.org:8080/cstar2/.

3.1.2.2 The Lockheed Martin SAVE Program

The Lockheed Martin Simulation Assessment Validation Environment[7] (SAVE) program integrates a broad spectrum of software tools to support aircraft manufacture. Within SAVE, CAD-based aircraft design data form a principal input to the development of the manufacturing process. The focus of the effort is, once again, on design and manufacturing data related to the structural elements of the aircraft. A series of demonstrations concentrate on the manufacture of aircraft structural elements.

CORBA is used to provide distributed object services accommodating the several software components written in different computer languages, with some of the components only available on particular hardware platforms. Initial efforts are being directed toward providing a single environment in which the individual manufacturing-related software components can operate. Each software component is to be available for selection for execution from any site within the system, with data exchange between components provided initially on a pairwise-basis as required. The system design makes provision for a series of program execution and data exchange protocols that lead in stepwise fashion to a full distributed object implementation.

The principal components in the SAVE system are planned to include:

- Dassault
  - CATIA 3D design
- Deneb
  - factory simulation
  - assembly simulation
  - ergonomic analysis
- IBM/Technomatix/VSA
  - assembly variability simulation
- Decision Dynamics
  - system dynamics simulation
- Pritsker
  - schedule simulation
- Cognition
  - cost modeling
- SAIC
  - risk management

Neither the architecture nor the design of the SAVE system includes any reference to use of agent-based software components. However, as a CORBA-based distributed object system, it would be relatively easy to include agent-based software components within the SAVE framework.

---

[7] The material on the SAVE program was developed from a series of Web pages beginning with http://skipper.mar.external.lmco.com/save/.

3.1.3 Recommendations with Respect to Virtual Manufacturing

Work that is underway in virtual manufacturing holds the promise of making significant contributions to related work in the logistics area. Our focus has been on the support of the development of maintenance procedures. Particular applications within the virtual manufacturing framework, such as product assembly and disassembly, have direct applicability to related areas in maintenance plan development and verification. As these tools come on-line in the virtual manufacturing area it should be possible to put them to use to support maintenance plan development. If maintenance procedure verification is addressed while still in the design process and supported by the tools of the design process itself, there is the potential to better address maintenance requirements to achieve product lifecycle cost savings.

The virtual manufacturing technology effort also has broader implications for the maintenance logistics arena. The virtual manufacturing effort relies entirely on product design data and makes new demands on the form that product data must take if virtual manufacturing goals are to be achieved. Not surprisingly, one of the principal requirements is in the area of semantic labeling. The requirement to be met is one of providing product data labeled so that software can be developed that can reason about the product design. Virtual manufacturing makes the same demands on product data that maintenance logistics is making. The particular demands for semantic labeling that will be made for virtual manufacturing will not be identical to those for maintenance logistics, but there can be expected to be significant overlap. Hence, the long range outlook is good—over time, product data should evolve to meet the semantic labeling requirements to support virtual manufacturing and maintenance logistics.

As we have seen, there are several large virtual manufacturing efforts underway that are based on the use of STEP to make the shared use of aircraft product data feasible. The SAVE and CSTAR programs both have been pursuing a series of test cases based on aircraft structural component data. They have not, unfortunately, chosen test cases that include avionics or weapons systems components that would have been of more immediate relevance to problems in the maintenance logistics area. In the short term, it is not likely that there will be a virtual manufacturing test bed in place that will readily support the development of agent-based software designed to facilitate maintenance logistics applications. Nevertheless, it will be worthwhile to track the SAVE program, as they are integrating a broad range of product manufacturing software tools and beginning to address the sharing of product data. As the virtual manufacturing programs mature, and in particular as they begin to address issues of component assembly for avionics and weapons systems, it will be important to revisit virtual manufacturing as an area that can provide input for the development of maintenance logistics procedures.

3.2 Aircraft Maintenance

The Mission Capable Parts (MICAP) system establishes policies and procedures for obtaining material items necessary to repair high priority USAF mission-essential equipment. The tracking and status reporting aspect of the MICAP system is one of the primary areas being addressed as part of the requirements analysis currently underway for

the Logistics Command and Control Information Support (LOCIS) program. Our initial goal in the current study was to understand the current MICAP process, the systems that support it, and how it is implemented and perceived by Logistics personnel. A series of interviews were conducted with maintenance and supply personnel at Hurlburt and Elgin Air Force Bases. These interviews were supplemented by additional data gathering at Wright-Patterson and Holloman Air Force Bases. The intent of the survey was to capture a top-level perspective of the current MICAP process. With this background in place, we then looked at how an agent-based system might play a role in improving the MICAP process. Volume 2 of this report provides a complete discussion of this portion of the Intelligent Agent Study. A brief overview is included here.

Our review identified two primary areas where state-of-the-art software and hardware technologies can be leveraged to help make significant improvements in the MICAP process. The first and most significant problem involves the *sourcing* and *tracking* of MICAP parts. There are a large number of potential on-base sources for any given MICAP part that must be checked by the Supply MICAP specialist before the off-base sourcing of parts is undertaken. Obtaining part availability information is frequently a time-consuming process, involving numerous phone calls or in-person visits to query each potential source. Potential off-base sources include lateral support from another base via the MICAP Asset Sourcing System (MASS), the depots, the Aerospace Maintenance and Regeneration Center (AMARC) or the Defense Reutilization and Marketing Service (DRMS). There is limited on-line system support for locating MICAP parts either on-base or off-base. The systems that support these functions are "stovepipe" systems that do not provide an integrated "one-stop" shopping capability for rapidly locating MICAP parts. Part tracking is important in maintaining visibility of a MICAP part from the time it is located and shipped from a source to the time the part is delivered to the maintenance crew needing it. Here again, the tracking must be accomplished via numerous phone calls with only limited on-line tracking capability.

The second problem area involves the accurate and efficient researching of MICAP part information. Current processes require the "stubby-pencil" transcription and duplicate input of lengthy and non-intuitive part and stock numbers. This information is typically obtained from the Technical Orders outlining repair procedures for the applicable aircraft. It is well recognized by personnel within the maintenance and supply process that it is easy to make errors in this process and personnel take great care in checking the accuracy of MICAP part information to preclude mistakes in the ordering of critical, high-dollar-value components. Supply personnel make the accuracy checks on part information twice during the typical MICAP sourcing and requisition process and only then is a MICAP request issued.

Today, the MICAP sourcing process, as played out by local supply and maintenance personnel, requires that they be in frequent telephone contact and that they make frequent calls to personnel at remote sites such as a depot or another base. The lack of system support for tracking parts means that the frequent updates necessary to monitor progress on MICAP problems must also be accomplished by person-to-person communication.

This is inefficient and time consuming for all parties and distracts the queried party from accomplishing on-going tasks.

Part tracking is the critical enabling technology that must be in place to enable progress in the MICAP process. With part tracking in place, a number of improvements in the MICAP process are possible, some of which can be addressed using agent-based systems. With many players working at local and remote sites, an agent-based system (based on a distributed object computing environment) can provide the connectivity to support the tasks associated with researching part information, part sourcing and tracking, and MICAP status reporting tasks. Support for locating and tracking parts will depend on network connectivity. If the connectivity is available, proactive agents can immediately be dispatched to multiple potential sources to check on the availability of a part as soon as it is identified. Active monitoring of parts in transit can be used to notify personnel of imminent or delayed arrivals. The process of preparing briefing materials on MICAP can be simplified and made available through frequent updating as necessary. Several of the players should have access to MICAP information via PDAs as they move among their workplaces. In contrast to the sourcing and tracking, support for part identification is a problem that can be addressed by a single agent or a small team of cooperating agents. Each of these agent-based services can play an important role in making the MICAP process work better.

3.3 Air Mobility Command

The Air Mobility Command's (AMC) computational system requirements[8] are currently addressed by a broad array of largely legacy systems with extensive interfaces upward to the Joint level and downward to the Unit level. This network of services is currently undergoing an extensive internal review. An important component of this review is the development of a corporate information vision that embodies operational, systems and technical architectures. The *operational* architecture provides "a description of the tasks, operational elements and information flows required to accomplish or support a warfighting function"; the *systems* architecture provides "a description, including graphics, of the systems and interconnections, providing for or supporting a warfighting function"; the *technical* architecture provides "a minimal set of rules governing the arrangement, interaction, and interdependence of the parts or elements of a system, whose purpose is to ensure that a conformant system satisfies a specified set of requirements."

The operational study identified ten broad categories of AMC support functions (e.g., provide material, conduct planning, perform analysis) and two broad categories of AMC execution functions—manage mission execution and conduct air mobility operations. The input products to and the output products from each function were identified. The application component of the systems architecture defined the functionality required by AMC. Fifty-five applications were identified, of which 27 are specific to AMC (e.g., aircrew and aircraft tasking, location capability, mission scheduling, cargo processing.)

---

[8] The information in this section was derived in part from a presentation entitled *Air Mobility Command Corporate Information Vision* by Lt Col Ed Kera (HQ AMC/SCTA) at Scott Air Force Base on April 16th and 17th, 1997. Subsequent quotes in this paragraph are from this presentation.

The information component of the systems architecture provides an integrated AMC logical data model currently containing 1457 data elements that will form the basis for a corporate data base. The analysis performed to date provides the basis for an enterprise-wide framework to meet the AMC C4 vision for global engagement. AMC has also identified a migration strategy to move from existing systems (e.g., GDSS, C2IPS, ADANS, CMARPS) to the new architectures.

An important component of the new AMC architecture will be the ability to import and exploit new technologies as they become available. The technologies that have been identified to date include: scheduling, allocation and decision-making; execution monitoring; visualization of information and operations forecast; rapid replanning and rescheduling; advance communications and security. The area of execution monitoring has been identified as one where agent technology is expected to find application.[9] Mission exception notification is a critical function within execution monitoring. Subject areas included under mission exception notification include sequence of events, cargo processing, passenger and patient status, aircraft position, aircraft support and ground support.

AS AMENDED

Current collaboration between AMC and DARPA is centered on the knowledge rover work[10] currently in process at Sumaria Systems, Inc., Dynamics Research Corp. and George Mason University. Mission and mission support areas being reviewed include air lift and air refueling. The project is examining the dynamic allocation and reallocation of AMC assets and the defining criteria, and internal and external events that will act as triggers to initiate the rapid identification and reprioritization of AMC resource utilization as changes occur.

Mission exception notification is a well chosen application for agent-based support. Whether labeled knowledge rovers, sentinels or intelligent agents, the actual distinctions among them were not addressed in the briefings. It is easy to imagine that many functional areas related to the execution of AMC operations are overseen by personnel required to monitor a broad range of inputs and, in turn, to provide inputs to a broad range of related downstream operations. Any given duty officer is likely to have a number of concurrent on-going tasks, each with time-critical decisions dependent on inputs from several sources. Some subsets of the tasks may be independent of one another, but others may have interlocking dependencies.

The complexity of such tasks and the requirement for situation awareness suggest that it is not appropriate to turn them over entirely to automation. However, identifying and monitoring the triggers that predict important events and collecting, assembling and collating related data are tasks well suited for agent-based support. Just as there are multi-

---

[9] The information related to mission exception notification was derived in part from a presentation entitled *Air Mobility Command Knowledge Rovers, Sentinals, and Intelligent Agents* by Maj Bill Fetech (AMC/SCTA) at Scott Air Force Base on April 17th, 1997 as part of the briefing lead by Lt Col Ed Kera.

[10] The information related to the AMC/DARPA knowledge rover application was derived in part from a presentation entitled *Air Mobility Command Knowledge Rovers, Sentinals, and Intelligent Agents* by Maj Bill Fetech (AMC/SCTA) at Scott Air Force Base on April 17th, 1997 as part of the briefing lead by Lt Col Ed Kera.

tasking demands on the duty officer, the agent or agents supporting the duty officer can be expected to be required to possess multi-tasking capabilities. Assigning individual tasks to a collection of individual agents fails to address the relations and interactions among on-going tasks that are, in many cases, critical to their successful outcome. It will be at least as important to avoid burdening the duty officer with the irritation of false alarms as to accurately identify situations requiring intervention. What might be a critical short supply in a particular munition might also be a matter that does not need to be addressed immediately because of the atypical character of that night's mission profile. The agents operating in this complex environment will need the capability to address a range of tasks and manage the implications of the interactions between them. These are important factors to be addressed in the design of the architecture supporting agents operating in a distributed system environment.

From the description of the AMC C4 vision it is clear that the new architectures define a very structured computing environment. The development of the integrated AMC logical data model was based on the careful analysis of the data requirements of existing systems to form the information component of the systems architecture. The applications component of the AMC systems architecture was similarly based on an analysis of the functionality required by AMC. In this analysis 57 applications with 27 specific to AMC were identified. A computing environment with this level of structure and organization contrasts sharply with the environments that are often used as the test-beds for research agent-based systems. Probably the most commonly used environment for agent research is the Internet with its nearly total lack of structure. The highly structured environment defined by the architectures for the AMC vision has the effect of relaxing requirements to be met by its agents. The logical data model will insure that the objects or data on which the agents will be operating will be unambiguous and well defined. Similarly, agents developed to operate in such a structured environment can be expected to have well defined and readily identified capabilities.

4. Agent-based Architectures

Recent research on agent-based systems has focused on the communication and cooperation among agents operating in distributed computing environments and on communications between system users and the agents that support them in the execution of their tasks. The development of multi-modal user interfaces represents an important application of agent-based technology (Moran et al., 1997). The new capabilities being demonstrated in agent-based systems reside in their potential to support widely dispersed users in their collaborative efforts to effectively manage evolving situations. In this section, we characterize potential application areas for an agent-based system, provide an outline of a system level architecture for an agent-based system, and discuss important attributes of the agents that might operate in this environment.

4.1 Application Characteristics for an Agent-based Architecture

Logistics operations cover a broad range of activities making considerable demands on the computer systems being employed to support them. It is important to look briefly at

the general computing environment that is evolving to support these operations. In this study we have focused primarily on the areas of deployment and maintenance, looking at agent-based computing tools that may find application there. In this section we look at the principal characteristics of these application areas that relate to agent-based architectures and establish some general requirements to be met by agent-based systems.

The work performed by the typical duty officer in many logistics operations has been identified as a set of tasks that would be well supported by an agent-based system. More precisely, it is the network of duty officers supporting a large scale deployment or coordinating maintenance and supply operations that might potentially be well supported by an agent-based system. In designing such a system, there are a large number of operating procedures that must be developed and supported, some of them to be executed by the duty officers and others by the system. Good communication among the duty officers is essential as are well defined user interfaces that seamlessly support the duty officers in the execution of their tasks. These are exactly the areas that have been focused on in the development of agent-based systems. The potential in agent-based systems is not so much in providing an "intelligent" agent to address a particularly complex task, but rather in supporting the collaborative efforts of a number of players working to together to manage rapidly evolving situations.

4.1.1 Distributed Computing Environments and Legacy Data Systems

Distributed object computing is getting significant attention within the Air Force today. As discussed above, a very significant effort is underway at the Air Mobility Command to establish a Corporate Knowledge Base for logistics operations. The objects for the distributed object computing environment are being defined, as are the functional areas that AMC must address. Agent-based computing fits naturally within this environment. Ideally, the agents in agent-based computing will be based on objects in a distributed object computing environment.

While the move to distributed object computing is underway, there will be a long period of time in which legacy systems are still in place, often providing important data sources to be used in distributed object or agent-based applications. The legacy systems may be client-server systems, older systems with ASCII-based user interfaces, or batch processing systems, any of which might be important as sources of data stored either in formal data bases or more loosely structured as program data. Object-oriented systems have found their first applications in just these areas, where it is important to obtain data from disparate sources and provide that data with the semantic labeling needed for further processing. The separate functionalities of individual legacy systems have been integrated to form federated systems to meet new user requirements. Agent-based approaches can be expected to play a similar role in gaining access to and integrating legacy systems.

4.1.2 Structured Computing Environments and Structured Data

The first and most obvious observation is that logistics systems to support maintenance and deployment are highly structured. While the subsystems and data objects available within the various legacy systems are evolving to meet new requirements, they have been

and will continue to be well defined. If data are required, the programs that develop that data will be structured to make that data available in a form suitable for use by the consumers of the data. Much of the effort in developing federated systems is devoted to tailoring data or functionality of one legacy system to make that data and functionality available in a larger framework. Much of the effort in the AMC Corporate Data Base effort has been devoted to developing single data objects that meet the requirements of closely related data objects defined in separate legacy systems.

In contrast with this highly structured environment of today's logistics systems, several agent-based research efforts have targeted data discovery or data mining as an important area to investigate. While the initial reaction might be to say that data mining is not an area that is likely to be appropriate for highly structured logistics systems, at least one example suggests otherwise. It is becoming more and more common for manufacturers and distributors to collaborate in making information about their products available via the Internet. This is information that might be used by logistics systems supporting maintenance and procurement operations. For local part procurement to support aircraft maintenance, data mining might simply take the form of an Internet search to find the source and best price for a given item. This area of electronic commerce has been an active area for research in agent-based systems (Chavez, Dreilinger, Guttman, & Maes, 1997; Tsvetovatyy, Gini, Mobasher, & Wieckowski, 1997).

4.1.3 Dynamic Situations

While logistics systems and data elements may be well structured, military situations can be expected to change quickly and dramatically. Network configurations will change to accommodate new deployments and a broad spectrum of situations will need to accommodate change along several dimensions. The response to a crisis situation or the transition from peacetime footing to wartime footing will dramatically alter the relationships among the data that the typical duty officer is evaluating. There are also several intermediate levels of preparedness between peacetime and wartime footings that must be addressed. Indeed, there may be concurrent situations, each requiring highly specialized services. Developing a proactive agent with the capability to monitor evolving situations and situation transitions will be very challenging. The multi-tasking nature of the duty officer's responsibilities is important. The duty officer will most likely be served by more than one agent. Each of these agents ought to be capable of multi-tasking behaviors and each will be required to manage the communication with its peers to accomplish individual and shared goals.

4.1.4 Serving the Duty Officer

Important posts in both the deployment and maintenance logistics areas are managed by duty officers heavily dependent on the timely flow of data from diverse sources. As we have seen, these personnel typically have a number of concurrent tasks in process at various stages of completion. They keep up-to-date by proactively seeking the information that they need, evaluating and making decisions based on that input, acting on those decisions and closing the loop by monitoring the outcomes of their actions. At the same time there is a reactive component to their activities. They must be responsive to

their superior officers, address requests from subordinates and respond to evolving situations. Supporting duty officers in the execution of the complex tasks that each accomplishes is an important goal that can be addressed using agent-based technologies.

There are several critical steps to be taken in using agent-based technologies to support the typical duty officer. A cognitive task analysis will provide the necessary insight into the task's data requirements, decision processes, execution, and execution monitoring and evaluation. In both the deployment and the maintenance regimes, task priorities driven by consumption rates can change radically with situation transitions. Timely access to accurate computer-based data is essential. Unless the capture of that data becomes a part of a process prior to the duty officer's requirement for it, agent-based software will not be able to support the duty officer that today relies on in-person conversations to gather the information necessary to critical decisions. Timeliness is important: there are systems supporting maintenance operations in which information is entered only at the end of each shift. In situations such as this, there is also reason to question the completeness and accuracy of the data. Logistics tasks, whether for deployment or maintenance, need to be structured so that data related to task progress are collected as that progress is made.

4.2 A Distributed OMAR-based Agent Architecture

The Operator Model Architecture (OMAR) (Freeman, 1997; Deutsch & Adams, 1995; Deutsch, Adams, Abrett, Cramer, & Feehrer, 1993) has functioned for several years as a simulation environment in which to develop human performance models, construct workplace models and evaluate scenarios including operating procedures for new equipment being considered for use in complex operating environments. The current implementation of OMAR's simulation component traces its roots to ACTORS (Agha, 1986; Hewitt, 1991), an early MIT research effort in building a distributed computing environment. Hence, it is not at all surprising that OMAR should be a candidate to support agent-based computing in a distributed object computing environment. While the original ACTORS implementation was not object oriented, the OMAR implementation is: OMAR objects, including agents and the goals and procedures that they execute, are defined as Simple Frame Language (SFL) concepts (Freeman, 1997; Deutsch et al., 1993) and instantiated as Lisp classes and class instances.

Application environments in which OMAR will operate have an impact on the architecture. The major factor will be the distributed operating environment in which Distributed OMAR might be called upon to operate. Today, the candidate distributed operating environments in which Distributed OMAR might operate include implementations of Object Management Group's CORBA and Defense Simulation and Modeling Office's (DMSO) High Level Architecture (HLA). While DMSO's HLA is primarily a simulation architecture, it provides many of the services of a distributed object system. There are several dimensions along which these environments differ significantly from one another. Entities in CORBA are primarily objects communicating through message passing. In HLA, entities will most often be objects that are responsive to events that take place in the simulation environment. While separate architectures might be developed to address these important differences, here we suggest that a single

architecture can span these approaches to agent communication. OMAR has always supported these two modes of communication and more recently, CORBA, while primarily a distributed object architecture, also defines an event service as a principal element of its architecture. At the level at which we will define the architecture for Distributed OMAR, it is expected that one architecture will be readily adapted for implementation in any of these distributed computing frameworks.

The basic elements of the architecture (see Figure 1) include site support for the operation of OMAR agents at host machines in the network, a naming service for distributed objects, communication services among agents at local and remote sites, agent access to application data, a framework for the development of agent behaviors at local and remote sites, and lastly, support for the development of application programs using Distributed OMAR for system building or as a resource within a larger environment.

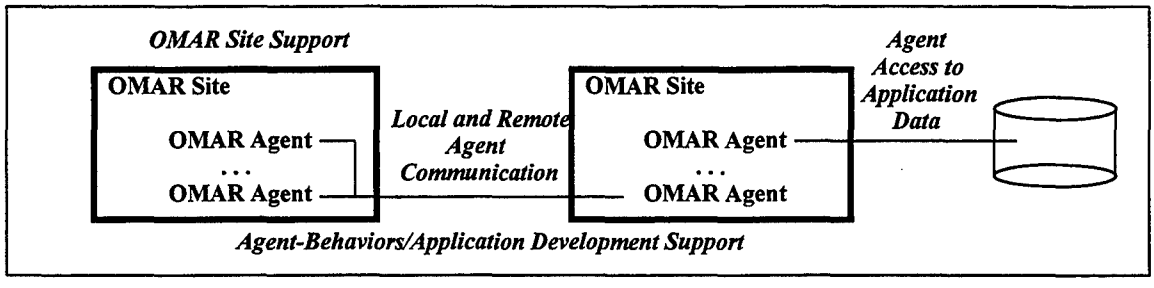

Figure 1. Architectural Components for Distributed OMAR

4.2.1 Distributed Object System Support

Distributed OMAR will be implemented within a distributed object system framework. But, that said, there are a number of implementation options and each may well have implications that the architecture for Distributed OMAR will have to accommodate. This is complicated by the fact that the offerings in distributed object frameworks are rapidly evolving, with a number of major software vendors playing key roles and many smaller players making innovative contributions. The major players (see Figure 2) are Sun with Java and the Remote Machine Interface; the Object Management Group, a consortium defining the specifications for CORBA with vendors offering CORBA-based software; and Microsoft with DCOM. However, DCOM precludes operation in a UNIX environment and is not a likely candidate environment. In the military simulation and modeling world guided by DMSO, compliance with the High Level Architecture will be required. Voyager, an agent-based system implemented in Java, from Object Space is one of the more interesting offerings from one of the smaller players. The situation is further complicated by the fact that there are a number of potential applications environments that Distributed OMAR might be called upon to play in, some likely to be based on CORBA, some others on HLA, and some on DCOM.

Recommended Change

Recommend Change

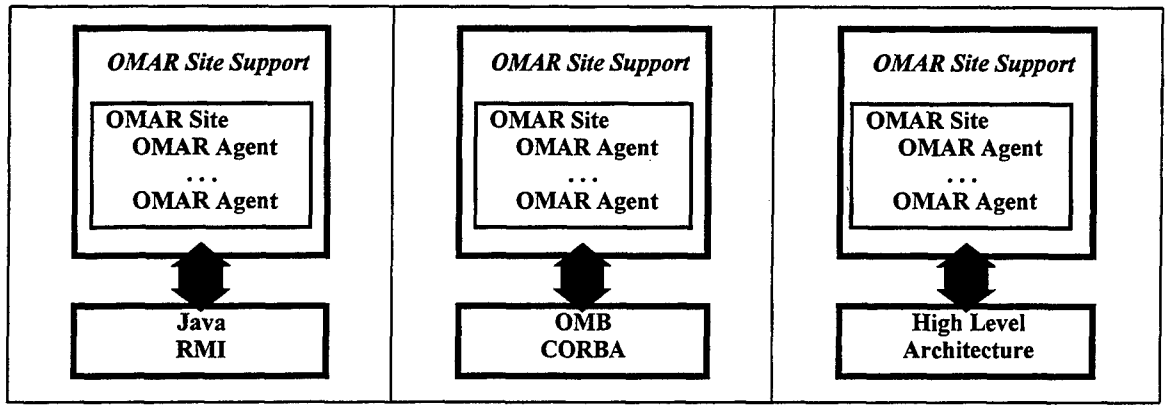

Figure 2. Implementation Options for Distributed Object Services

The most straightforward approach to creating a Distributed OMAR would be to supplement the existing OMAR framework with Java-based extensions. As in any approach, the Lisp-based OMAR sites would provide a bridge between Lisp and Java. Network connectivity among OMAR sites that supports message passing, event processing, and object naming services would be developed in Java. The communication code could be developed directly in Java or it could be based on the Java Remote Method Invocation (RMI). It is particularly interesting that Object Space's Voyager, while developed in Java, does not make use of the RMI. Object Space claims to provide greater functionality and better performance in a comparison with their Voyager system with RMI[11].

The requirement to use Distributed OMAR as part of on-going military applications impacts the network connectivity implementation decision. In several application areas CORBA is already in use or is likely to be the required vehicle for network connectivity, while in the simulation and modeling area, the HLA will be required and can be expected to supply many of these connectivity services. CORBA has been the middleware substrate of choice for a number of military systems that include legacy system components, legacy data bases or legacy simulators as system components. The major CORBA feature is the capability to accommodate system components developed in different computer languages. Some of these systems are already using Lisp-based components. The High Level Architecture, discussed in more detail below, is a Defense Simulation and Modeling Office defined standard architecture to support the integration and coordinated execution of military simulators and models. Predecessor systems (Deutsch, 1993) to OMAR played an important role in the early development of SIMNET Semi-automated Forces. OMAR now has the potential to play a significant role in a variety of the HLA simulation environments, both for agent-based supporting roles and for modeling human players or military units.

In summary, distributed object system support is expected to be an implementation level issue rather than an architecture issue. Object-oriented message passing and event

[11] The Object Space comparison between Voyager and RMI is available from the Object Space Web site at http://www.objectspace.com.

processing that were once the important features of two distinct architectures have been combined within the current OMAR system, and more importantly, within CORBA. A single agent-based system architecture can now accommodate the candidate system support environments. The basic networking and distributed object services required by Distributed OMAR can be covered in each of the potential implementation systems. The choice for the first implementation will be driven by the application environment in which Distributed OMAR will play a role.

4.2.2 Site Support for OMAR Agents

The agent framework forming the environment for Distributed OMAR at a site, or host machine, is a Lisp image. There is no reason not to continue to use Franz's Allegro Common Lisp as used in the current implementation of OMAR. OMAR is presently operational on Sun and SGI machines. Franz has recently released a "UNIX version" of Allegro Common Lisp for Windows NT and there is an earlier version of their Lisp that runs under Windows 95. Several of the basic site services required by Distributed OMAR are provided by OMAR running in today's Lisp image. These include the ability to create and destroy agents, and support for communication among local agents. Extensions will include access to the object naming service that should be provided by the distributed computing platform. In CORBA, it would take the form of the CORBA Object Naming Service[12] (Orfali et al., 1996). Additional requirements include facilities to support communication between local OMAR agents and agents at remote sites, and OMAR agent access to application data and services provided by heterogeneous agents.

4.2.3 Agent-Agent Communication

The principal form of communication among agents in OMAR, particularly those that will reside at remote sites, is signal passing. In OMAR today, an agent may, via one or more of its running procedures, subscribe to a signal. That is, the agent makes a request to be notified of the occurrence of a particular event announced as a broadcast signal that usually includes several data elements. The request may specify a test to be performed on one or more of the data elements included with the signal. The compiler generates a closure for the function to be executed by the agent upon a successful outcome of the specified test condition. At any given time there may be one or more agents, each with one or more procedures enqueued on the occurrence of any given signal. It may also be the case that a signal occurs that is not attended by any agent.

Signal processing in OMAR maps closely to the "push model" as defined for the Event Service[13] (Orfali, et al., 1996) in CORBA. Given a CORBA implementation that included the Event Service, there are just a few steps to be accomplished to implement the OMAR signal processing capability in a distributed environment. OMAR signals currently defined as lists would be reimplemented as objects—a simple and straightforward

---

[12] Detailed information on the CORBA Naming Service is available at the Web site http://www.omg.org/library/corbserv.htm.

[13] Detailed information on the CORBA Event Service is available at the Web site http://www.omg.org/library/corbserv.htm.

change. Next, the current OMAR capability to propagate signals to and receive signals from outside the Lisp image would be adapted to interface to the event services of the selected distributed object system. In CORBA, interest in an event persists until it is turned off. In OMAR, enqueuing on a signal persists only until the first successful application of the test condition or the first occurrence of the signal type if no test is specified. Minor coding would be necessary to accommodate this difference.

Within OMAR, agents are implemented as objects and also use traditional object-oriented message passing as a mode of communication. The message passing that currently operates locally in OMAR would be extended to operate in the distributed computing environment. These are implementation issues rather than architecture issues.

4.2.4 Homogeneous and Heterogeneous Agents

The strongest case for supporting homogeneous agent-based systems has been made by Genesereth and Ketchpel (1994). They argue for syntactic and semantic homogeneity using the Agent Communication Language as the basis for agent communication to enforce the homogeneity. Ginsberg (1991) argued strongly against this position, basing his argument on his view that knowledge representation was and is an immature science, and therefore it is premature to commit to a single knowledge representation schema pointing to already perceived shortcomings in the proposed standard. Ginsberg's case is supported by the significant growth that has taken place in the development of agent-based systems. The knowledge representation methods utilized are anything but homogeneous and as a practical matter homogeneity is not to be hoped for in the near future.

The architecture for Distributed OMAR addresses agent heterogeneity on three levels: syntactic, control and semantic heterogeneity as identified by Bird (1993). From its earliest days, OMAR knowledge representation (Freeman, 1997; Deutsch et al., 1993) was based on a frame language, the Simple Frame Language (SFL), and a procedural language, the Simulation Core language (SCORE). More recently, a rule-based language has been included in OMAR. At the syntactic and control levels, OMAR will continue to be an open system allowing the inclusion of new knowledge representation schemes and reasoning mechanisms.

At the semantic level there are reasons for both homogeneity and heterogeneity. The case for homogeneity is strongest when addressing communication among OMAR agents. OMAR agents should have a common semantics as a basis for their communication. This is not enforced by the architecture, but is encouraged as good programming practice. The argument for heterogeneity is based on the pragmatics of the growth in agent-based systems. It is highly unlikely that OMAR will be the only source of agent-based software in any given large Air Force system. Such systems are likely to be populated by agents from several sources (see Figure 3) and, while CORBA may address syntactic heterogeneity, semantic heterogeneity will have to be addressed by OMAR agents and non-OMAR agents populating a given system. The functionality provided by additional agents in a system should be viewed as a resource, while semantic heterogeneity is an obstacle to be overcome in taking advantage of that functionality.

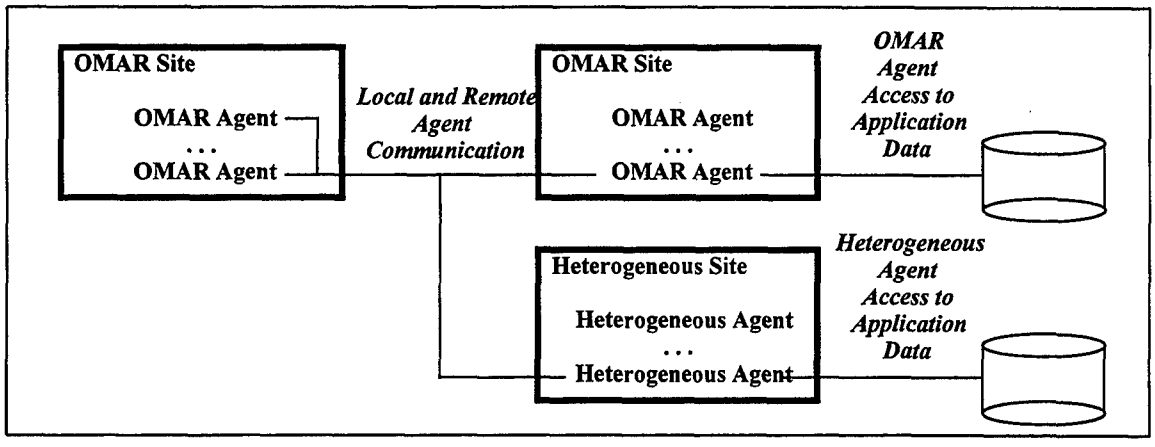

Figure 3. Distributed OMAR and Heterogeneous Agents

4.2.5 Agent Access to Application Data

Access to application data can be expected to take several forms. In the case where Distributed OMAR will be a new addition to the existing system, the architectural requirements for OMAR will be dictated to some extent by the architecture of the particular application system. In the most forward looking perspective, Distributed OMAR would be operating in a distributed object environment. A CORBA environment is a very likely possibility. The Defense Modeling and Simulation Office High Level Architecture—discussed in more detail below—is another form that the distributed object environment might take. At the other end of the spectrum, data required by an OMAR agent might reside in a legacy data base, most likely as part of a large legacy system.

Over the last several years there have been a number of systems built around pre-existing legacy system components. It has typically been the case that the components have included legacy data bases and large application software subsystems with individual components implemented in different computer languages. These systems are often referred to as *federated* systems. The middleware to provide interoperability is usually CORBA. And hence, it is CORBA that would provide the interface that Distributed OMAR would use to operate in one of these federated environments. The architectures for these systems are well established. At the architectural level there are not likely to be demands that will not be met with the evolving Distributed OMAR architecture. Indeed, at the implementation level, some of the essential software building blocks of current systems may well be available for reuse.

As the legacy systems are replaced by new systems based on distributed object architectures, these architectures can be expected to closely resemble the architecture being specified for Distributed OMAR. Such systems are expected to rely heavily on traditional message passing and OMAR will access objects and services in these systems via message passing. This will be in contrast to the signal-based communication that OMAR agents are expected to use predominantly for their own inter-agent communication. Message passing and signal-based communication will both be required components for OMAR in a distributed object system environment. OMAR's heavy reliance on signal-based communication, while not typical of most emerging distributed

object systems, is closely related to the communication style found in the High Level Architecture environment. New distributed object systems can also be expected to include agent-based components as the service providers. It will be to the OMAR agent's advantage to make use of these service providers by developing interfaces to these providers.

4.2.6 Application System Services

Two distinct approaches to the development of application programs are envisioned. In the first, Distributed OMAR itself would be used as the basis for building the application. In the second approach, the services provided by OMAR agents would be used as part of the development of a larger system. A set of core OMAR services will be needed to support both approaches.

An important element of the core services is support for user interface development. The current OMAR system provides data analysis displays for the evaluation of OMAR simulation runs, as well as an extensive set of software development and debugging tools. The goal of making these software tools available in the Distributed OMAR environment forms a sub-goal of the larger effort to support user interface development. A Web-based interface to OMAR will provide the most straightforward approach to gaining remote access to OMAR. Today, there is a large body of Lisp code that supports the elements of the OMAR analysis and software development tool set. The code that generates the displays and manages the displays themselves is a similarly large body of Common Lisp Interface Manager (CLIM) code. The new architecture will formalize the boundary between the current Lisp application code and CLIM code, establishing intermediate display management objects to handle what we may informally refer to as display data. These intermediate data structures will be capable of representing the broad range of user interface elements that make up today's user interface to OMAR. The two goals of this aspect of the architecture are to make it possible to write the display code in a non-Lisp language such as Java and to support network connectivity to provide remote access to OMAR (see Figure 4).

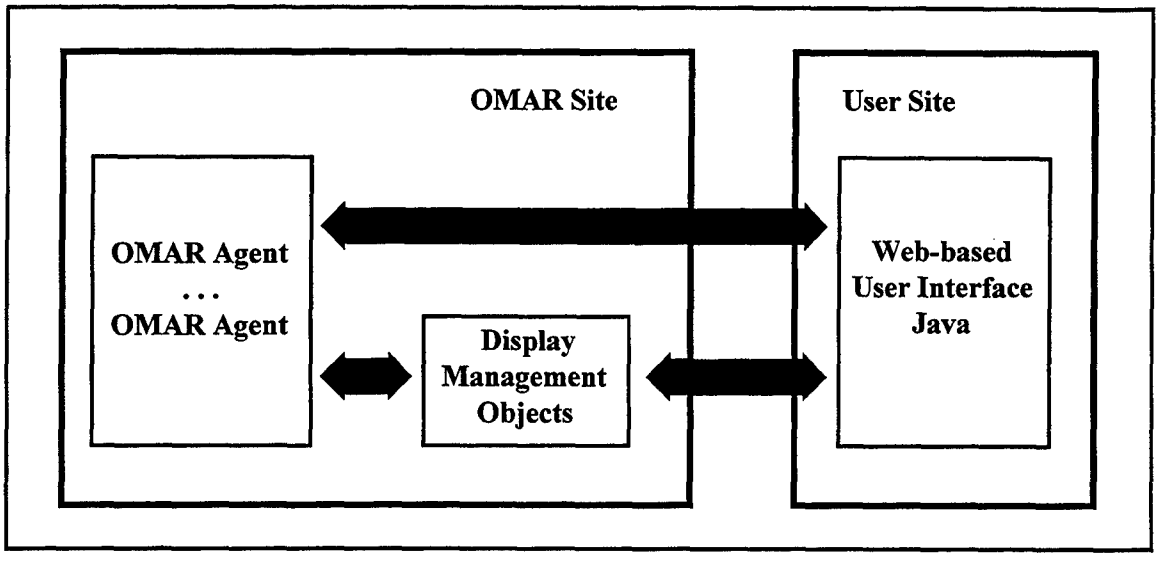

Figure 4. OMAR Web-based User Interface

The interface described in the previous paragraph provides access to OMAR data that have been highly processed for creating user interface displays. It will also be necessary to access OMAR objects before this display oriented processing has been executed. This more direct access to OMAR objects will facilitate more direct control of OMAR services, as well as providing the ability to develop new user interface elements based on OMAR data using a language such as Java running at the user's site.

4.2.7 High Level Architecture Compliance

The Defense Modeling and Simulation Office (DMSO) High Level Architecture (HLA) initiative has the goal of facilitating the interoperability of simulation components and models, and further insuring their interpretability with C4I systems. HLA is addressing, on a large scale, what OMAR has been working toward on a smaller scale: the integration of real-time operations with simulation components. OMAR has many capabilities that may become powerful tools in this integrated operating environment and it is important that the new Distributed OMAR architecture address requirements for HLA compliance.

HLA will, in fact, provide several important services when OMAR agents operate in that environment. Within an HLA framework, Distributed OMAR might function as a federation with particular OMAR agents operating as HLA federates. An HLA data model would specify the form taken by the objects exchanged by OMAR agents. The HLA Real-time Infrastructure (RTI) will provide runtime services for OMAR-based HLA federates. When the simulation environment includes active C2 systems with human players, HLA provides the interface linking the real time data of the C2 system to the HLA framework, hence providing direct access to this data for OMAR agents. An OMAR agent, acting as an HLA federate, could potentially sign up to receive real-time C2 system-generated objects just as it would to receive OMAR agent-generated objects.

The HLA interface services are logically structured into several categories (see Figure 5): federation management; declaration management; object management; ownership

management; time management; and data distribution management. Federation management services will be available to enable the OMAR agents forming a federation to join a simulation environment. Declaration management, object management and data distribution management are the means by which object types are defined, and their ownership, authoring and distribution are managed at runtime.

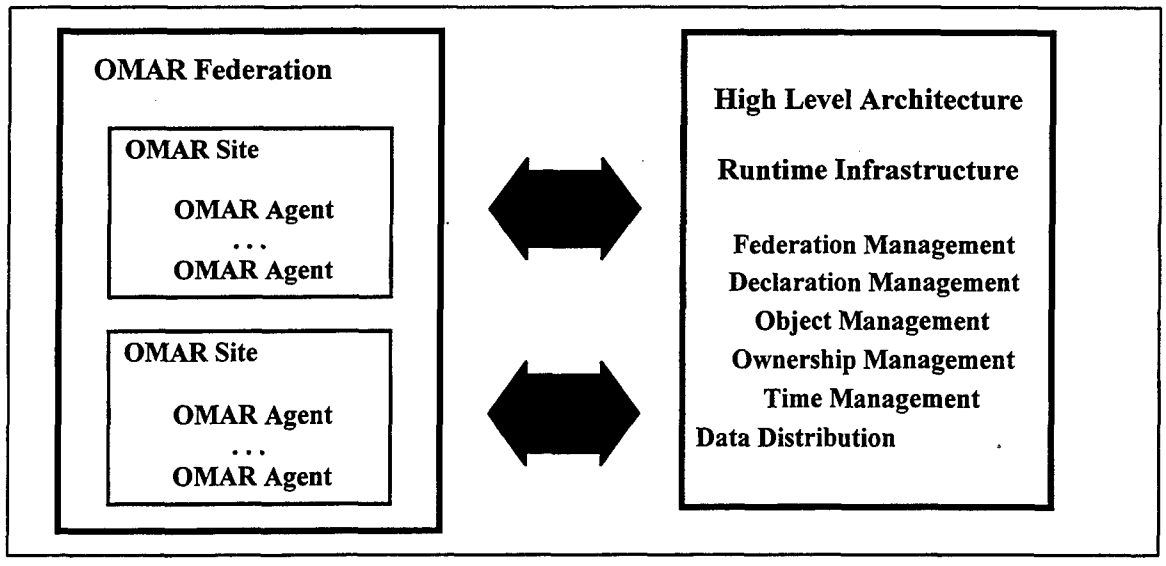

Figure 5. High Level Architecture Compliance

At the architectural level, OMAR compliance with HLA should be quite straightforward. A network of OMAR sites would form a federation with the agents operating as federates at the OMAR sites. The federation would be established via calls to the HLA RTI. In the Distributed OMAR architecture the principal means of communication among agents is envisioned to be OMAR signals. In the HLA world the OMAR signals would become and be defined as Simulation Object Models (SOM). Using services at the OMAR sites, an OMAR agent would sign up to receive particular signal types. The RTI also provides time management services for the simulation environment. The interface to the HLA RTI for time management services would be via the event queue at each OMAR site in the network.

Beneath the architecture level, OMAR access to HLA services will require a software bridge to span implementation language differences. The current C++ implementation of HLA and Lisp implementation of OMAR will require the development of several inter-language services. The foreign function calling facility with the Lisp environment will provide the means to access HLA RTI services from OMAR sites. Additional services will be required to translate instances of simulation object models between Lisp and C++ and C++ and Lisp. A Java version of the HLA RTI is under development. Since OMAR code supporting communication between Lisp and Java is already under development there would less effort involved in building the interface to the Java version of the HLA RTI.

4.3 OMAR Agent Architecture

Research on agents and agent-based system has been underway for several years (Lesser, 1991). More recently, object-oriented systems and client-server systems have become dominant themes in computing environments. These two themes are merging (see Figure 6) in the development of distributed object computing environments. One goal is the development of clusters of objects and their methods to form software *components*. In a distributed object computing environment these software components are expected to package capabilities suitable for reuse in varied environments. In a newly evolving perspective, an agent may be viewed as a special case of an object and an agent-based system may be viewed as a special case of an object-oriented system. And in a distributed object computing environment, an agent may be viewed essentially as an object enhanced with few or many special capabilities depending on the particular definition of an agent. In this section we examine agent capabilities that establish them as candidates to implement software components in a distributed object computing environment. In particular, we look at the attributes of OMAR agents that can be expected to contribute most to this development.

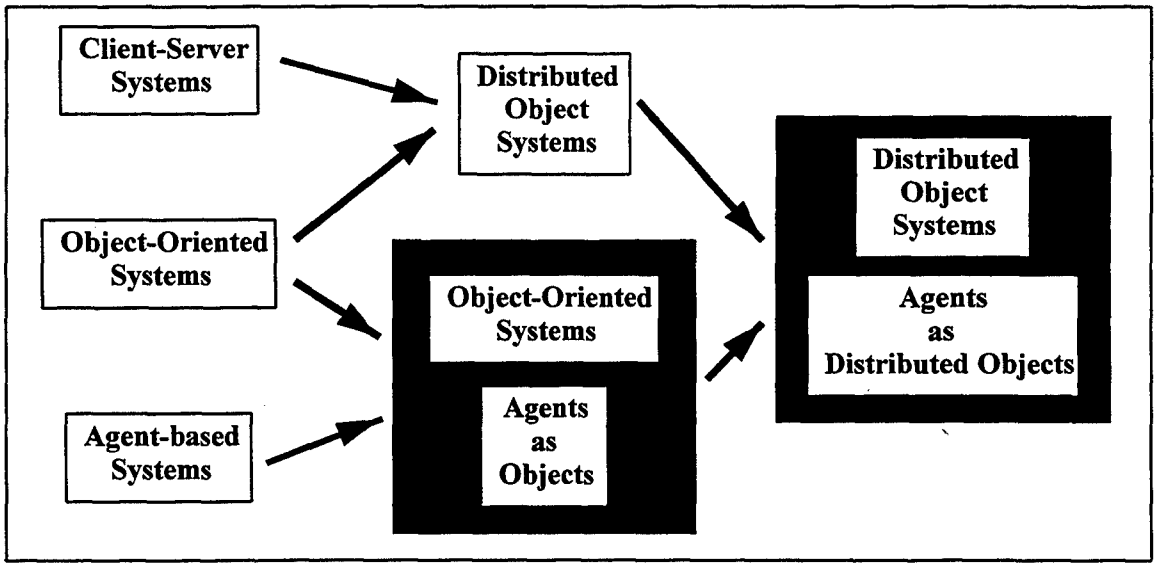

Figure 6. Agent Roles in Evolving Computer Environments

The agents in systems such as Object Space's Voyager and IBM's Aglets are actually very simple. They are object-like in that they can respond to messages and they can send messages to request services, and while they are mobile, they are much restricted in their ability to carry a copy of their state and processes with them. The architectures to support ACL-based agents and SRI's OAA are more complex, particularly in their use of facilitators. Each of these systems delegates important functions to facilitators. The facilitators require that agents register their capabilities and then take on the task of matching agent service requests with agent providers. In an implementation, the facilitators might well be agents themselves. We have also briefly touched on the research on agent-based systems related to the skills of cooperation and negotiation. Hence, the demands on agents can be quite complex.

As a software development environment, OMAR was designed to support the development of human performance models. It provides a set of software tools in the form of representation languages, and language editors and browsers to create models of human operators and their interactions with complex computer-based systems. Much of the modeling has addressed studies in commercial air traffic control. Human performance models have been developed both for air traffic controllers and the flight deck officers of commercial aircraft. The models have principally examined communication between the flight deck officers and air traffic controllers and human-machine interactions taking place on the flight deck and at the air traffic controller's workplace. Each of the human players that is modeled typically has several tasks in process and interruptions are a common occurrence. In the scenarios developed in this modeling environment analysis tools play an important role. They are needed for performance evaluation and they are also essential to support the development of the scenarios and models themselves.

The human capabilities and attributes that we try to emulate in human performance models are very much like those that we would like to have in the agents in agent-based systems. The agents should be capable of goal-directed proactive behaviors, while at the same time, they should be capable of reacting properly to the sequence of events evolving in complex situations. In supporting these behaviors the agents require communication skills in order to draw on the capabilities of other agents as resources, and they must be capable of responding to requests made by other consumers in the system. The consumers in this environment may be other agents or the users or operators of the system itself. In the following sections, we examine the features provided by the OMAR languages for developing these capabilities.

4.3.1 Proactive and Reactive Behaviors

OMAR agents are capable of proactive behaviors, that is, they pursue objectives and they maintain an agenda of things to do in order to accomplish those objectives. The objectives are expressed as *goals* and the actions to accomplish those goals are organized in a *plan.* The plan to accomplish a goal may partition the goal into sub-goals. Actions to accomplish the goal or its sub-goals are expressed as *procedures*. Goals and procedures are typically defined to operate in an environment in which failures are anticipated and alternate paths to successful execution are prepared. In building a good plan, the paths that might lead to failure are studied and triggers are identified to anticipate situations leading to the failure of a portion of a plan. Multiple plans may need to be available to achieve a goal; each may be available to operate from a different set of initial conditions. And lastly, in a complex environment a plan in process to accomplish a goal may succeed or fail independently of the actions underway to achieve the goal.

Within OMAR, the simulation core (SCORE) language is used to define the goals, plans and procedures of OMAR agents. Goals and procedures are expressed using the *defgoal* and *defproc* forms. The plan for a goal is expressed through the sub-goals and procedures that the goal invokes through subroutine calls. The called subroutine may be another goal or a procedure. The *defgoal* form includes an argument provided for specifying the initial conditions required by the plan for the goal. It also provides arguments to specify the

exogenous success or failure conditions (as described in the previous paragraph) that might prematurely terminate actions to address a goal.

As we have seen, agent-based systems provide a set of site services at each node in the network at which their agents operate. A major feature of an OMAR site, the operating environment for OMAR, is a simulator in which an agent's goals and plans are played out. The proactive behaviors of an OMAR agent are governed by the several goals, typically, being actively pursued.

On the surface, the reactive behaviors of an agent, often driven by the simple occurrence of an event, appear far simpler than an agent's proactive behaviors. Within OMAR, events are expressed as a *signal* having a type and attribute-value pairs. From an OMAR agent's perspective, signals must be announced in a form accessible to the agent: the agent must have a procedure in process that is enqueued on the occurrence of the signal. Often, it is convenient to broadly define signals and allow the agent to examine variables associated with the signal and then decide to process or not process the signal. One or more agents may respond to a given signal, each using different criteria and examining different signal attributes in making their respective decisions to process or not process the signal. This example also exhibits the distinct differences between signal processing and subroutine calling or object-oriented message passing. A subroutine call or a message sent to an object invokes a single respondent; a signal may be processed by any number of procedures active on one or more agents, or perhaps, none at all. A called subroutine always returns control to its caller; an agent's procedure that processes a signal is operating in a thread independent of the source of the signal. Agents process a signal in their own code thread and do not return control or values to the initiator of the signal. Unless a particular effort is made to make it known, the initiator of a signal will not know which procedures took action as a response to the signal.

Reactive behaviors addressing single events, while they may be processed by more than one OMAR agent, are still quite straightforward. The development of agents to address more complex situations is made possible by two important OMAR capabilities. First, agent behaviors are seldom designed to be purely reactive. Even in simple cases, the occurrence of a particular event relevant to an agent will need to be addressed by that agent in the context of some goal to be achieved. The event might mark the successful achievement of the goal or it might mark a condition that indicates the failure of one tactic to achieve the goal and the need to choose another approach. Agent behaviors are often required to be a combination of proactive and reactive components. The processing of a signal marking an event plays an important role in the plans developed for the goals of OMAR agents.

The second source of complexity to be addressed by OMAR agents is the need to deal with sequences of events that may or may not be well ordered. The response of an OMAR agent to evolving situations is developed through the processing of the chains of events made available to the agent as signals. An agent's sensitivity to a given signal is set up by an agent's procedure enqueueing on that signal. The response to that signal may then be to set up the same procedure or another procedure in anticipation of the recurrence of the same signal, another particular signal, or a combination of the two. A

signal processed in one situation may be ignored in another situation. An agent might well have two evaluation processes going on concurrently, each with the capability to shut down the other and determine the response: a winner-take-all strategy.

The SCORE language provides four forms for managing an agent's signal processing. An agent generates a signal using the *signal-event* form to define the signal type and its arguments. The *asynch-wait* form is the simplest form used to enqueue on a signal specifying the signal type and the argument values required. Upon enqueueing on a signal, that thread of the agent's processing is suspended until the occurrence of a signal with an exact match on the signal type and argument values. The *with-signal* form provides more flexibility in selecting a signal to process. In using this form, an agent gains access to each signal of the specified type and may then examine any or all of the signal's arguments for specific values before deciding to process the signal. If, through the use of the test clause, the agent elects not to process the signal, the *with-signal* form remains in effect awaiting the next occurrence of the specified signal type. The fourth form, *with-multiple-signals*, as its name suggests, provides the capability for processing several signals that may not be well-ordered. As with the form *with-signal*, a test clause may be specified for each signal specified in the form. Time-out conditions may be associated with each signal and the occurrence of a signal may be tagged as optional within the form. Hence, two or more signals may be required to occur in a given time frame if further processing is to take place in a given procedure. An optional signal may provide additional information that could support subsequent processing.

4.3.2 Multi-task Performance

As we have seen, OMAR agents are proactive as is expressed in their plans and procedures to achieve their goals. And we have noted that an agent can have more than one active goal. That is, the agents are capable of multi-tasking. Some of an agent's goals can be unrelated and independent of one another, while others may be sub-goals within a plan with explicit dependencies among them. One dimension of the dependencies is the order in which sub-goals or procedures within a plan execute. They may be required to execute sequentially with one procedure strictly following another or they may be allowed to execute in parallel. In OMAR, as a modeling environment, it is the OMAR simulator that provides the emulation of parallel procedure execution. In real-world applications, execution of the OMAR simulator in real-time enables OMAR agents to pursue sub-goals in parallel. Termination conditions for sub-goals being pursued in parallel are important. In the winner-take-all case outlined above, the procedure that completes first has met the local objective and the concurrent procedures pursuing that objective should be shut down. In other cases, all procedures operating in parallel may be required to complete before subsequent procedures in the thread can be initiated.

As outlined in the multi-tasking discussion so far, there has been no explicit attribution of success or failure to a goal or a procedure—although success and failure clauses in the *defgoal* form were outlined earlier. By default, the completion of a goal or a procedure is successful. However, a procedure may declare itself to have failed and it may force the termination of another goal or procedure attributing success or failure to it. In the case of

the forced termination of a goal or procedure, two aspects of the termination are covered for OMAR agents. In the first case, the procedure forced to terminate, whether successfully or unsuccessfully, may need the opportunity to "clean up" before it actually completes execution. Provisions are available to specify the necessary clean-up steps on either successful or failed termination. The second case addresses the behavior of the procedure that invoked the procedure that has been forced to terminate. When the procedure that is forced to terminate does so successfully, no special action is required by the calling procedure, although additional steps may be specified for the calling procedure to execute. However, the calling procedure of a failed subprocedure will itself fail unless that failure is trapped. Goal and procedure failures will propagate upward unless they are explicitly trapped.

Goal and procedure conflicts are another essential aspect of multi-tasking behaviors for OMAR agents. The execution of one procedure may preclude the execution of another procedure. Priorities for goals and procedures, the basis for resolving these conflicts, are computed dynamically. The form of the priority calculation and its input parameters are specified by the developers of the agent's goals and procedures. Importantly, OMAR provides support for the basic fact that not all procedures conflict with all other procedures. Goals and procedures are class members and one important aspect, class membership, covers procedure conflicts. Procedures within a class may conflict with one another, or procedures in one class may conflict with procedures in one or more other classes. When a new procedure vying to execute has class membership such that it conflicts with an executing procedure, the priorities for both procedures are computed. If the new procedure has sufficient priority, it will begin execution and the executing procedure will be suspended. If it lacks sufficient priority it will be suspended until the executing procedure completes or until the balance of priorities shifts in its favor.

Several key SCORE forms provide the capability to define these multi-tasking behaviors for OMAR agents. Parallel execution of an agent's goals and procedures is specified using the forms *join, race* and *satisfy*. Procedure invocations enclosed in a *join* form must each execute to completion before subsequent forms are evaluated. The winner-take-all case is supported by the *race* form—the first procedure to complete causes all of its siblings to be terminated and initiates execution of subsequent forms. The *satisfy* form is a slight variation on the *race* form. Procedures executing in parallel within this form continue to execute until the first of them completes successfully. The *succeed* and *fail* forms, each without an argument, cause the executing procedure to terminate in success or failure respectively. When used with an argument, the argument specifies the procedure that is to be terminated. The *on-succeed*, *on-fail* and *on-suspend* forms are available to specify code to be executed when a procedure succeeds, fails or is suspended due to a conflict with a procedure with a higher priority. The important SCORE form, *fail-handler,* is used to trap and handle the failure of a subprocedure and thus avoid the continued upward propagation of the failure. As outlined above, goal and procedure class membership and priority are used as the basis for managing conflicts among goals and procedures. The SCORE forms for defining goals and procedures, *defgoal* and *defproc*, are used to specify the SFL concept membership for the goal or procedure, and this in turn specifies the class membership. The *conflict-with?* Method, defined as the goal or

procedure classes among which there are conflicts, is the basis for managing the priority-based suspension and resumption of conflicting goals and procedures.

5. Agent-based Systems: Costs, Benefits and Risks

The logistics arena is one in which advanced system development is making rapid progress. In assessing the risks associated with employing agent and agent-based systems for logistics applications we first looked at these recent developments in logistics systems, then at the viability of including agents and agent-based systems in the new systems under development. In the final section we examine the factors that limit the risks in pursuing the application of agent-based technology for logistics system development.

5.1 Advanced Department of Defense System Developments in Logistics

Within the Department of Defense, the logistics arena has been a very important application area for the development and deployment of advanced computer systems over the last several years. The Dynamic Analysis and Replanning Tool (DART) and Analysis of Mobility Platform (AMP) are two systems that have been developed and deployed in the last few years. Rapid prototyping played an important role in the development of each of these systems. The Advanced Logistics Program (ALP) is presently in the design phase and the Air Mobility Command is at work on a corporate data base for logistics operations.

Legacy systems, simulation models and data bases play important roles in each of these frameworks. Each of these systems is heterogeneous, with components written in different computer languages and running on a wide variety of hardware and software configurations. Each is a distributed system with clusters of components operating or designed to operate at a network of remote sites. AMP, as a simulation system, is currently being upgraded to run under the DMSO High Level Architecture. ALP will almost certainly use CORBA as the middleware to support its object-oriented distributed design. The work on the AMC corporate data base is predicated on a distributed object computing environment.

There is a very clear trend in the design of the logistics systems within the Department of Defense. Each of the advanced systems is a heterogeneous distributed system. The newest systems designs, the AMC Corporate Data Base and ALP, are distributed object systems.

5.2 Agent-based Systems and Advanced Development in Logistics

As we have seen, it is not easy to distinguish agent-based systems from object-oriented or distributed object systems. Those who have been most successful in sharply defining agent-based systems (Genesereth & Ketchpel, 1994) have done it at the cost of restricting the definition of agents to those object-like software entities using a specific suite of communication and knowledge representation languages (ACL, KQML, and KIF). Others (Ginsberg, 1991) have strongly argued that it is premature to close off diversity in this important research area. Given the broad range of research activities in agent-based systems in academia, industry and within the military, it is apparent that there is much

new ground to cover. The heterogeneous nature of the on-going and new efforts, both in the research arena and in systems development, should be able to accommodate this diversity.

Current work on the AMC Corporate Data Base and in ALP are representative of development programs in which agent-based approaches to logistics applications can be expected to play important roles. Each of these new environments will be heterogeneous in its computational components. They can each be expected to be distributed object systems and it is reasonable to expect that solutions to diverse logistics problems will be attempted by individual groups using diverse approaches to their solution. The range of problems to be addressed in logistics applications is very broad, with several of these problems likely to be addressed using agent-based systems, and it is premature to insist on a single agent-based framework to address these diverse problems.

5.3 Costs, Benefits and Risks

An assessment of costs, benefits and risks plays an important role in the evaluation of a new technology. When a new technology is revolutionary, as agent-based systems are sometimes portrayed, an accurate assessment can be very difficult to establish precisely. In the current assessment of agent-based software, the new technology is viewed as evolutionary rather than revolutionary. It has its roots in object-oriented programming and more recent developments fit within the broader landscape of distributed object systems. The software industry has been hard at work in recent years supporting object-oriented programming and is now hard at work on the middleware to make distributed object systems feasible. Several software vendors provide implementations of CORBA, Sun is promoting Java and RMI, and DCOM is the analogous Microsoft product. In academic settings and the research laboratories of commercial companies software to support the development of agent-based systems is an active endeavor. Some commercial agent-based products are just becoming available, while other nascent commercial products, while not yet fully supported, are available for testing.

Within the Department of Defense, and more particularly within the logistics community, object-oriented software technologies have played important roles in recently fielded systems such as DART and AMP. The newer distributed object system technologies are important in the AMC Corporate Data Base and ALP. As agent-based technologies are developed they will operate in the new distributed object frameworks under development. Their development will lean directly on these broader initiatives in software technology development. Particular capabilities such as naming and event services and inter-agent communication that were once capabilities that agent-based systems had to provide for themselves are now services that the middleware for distributed object systems is providing. This can have a significant impact in reducing both costs and risks in agent-based system development.

It is also important to point out that agent-based systems are not the "magic bullet" long sought by the software industry. The designers and builders of the applications still bear the burden of creating the problem solutions. But agent-based systems do provide system designers with a new perspective from which to approach system design. Agents by their

very nature are designed to communicate with and serve not only other agents, but also system users. They have the potential to be good building blocks for creating systems to support real-time operations, such as duty officers in their complex, interconnected world where they must assimilate a broad range of inputs and stay abreast of and interact with a wide range of evolving situations.

6. Acronyms

ACL	Agent Communication Language
ADANS	AMC Deployment Analysis System
ALP	Advanced Logistics Program
AMARC	Aerospace Maintenance and Regeneration Center
AMC	Air Mobility Command
AMP	Analysis of Mobility Platform
C2IPS	C2 Information Processing System
CAD	Computer Aided Design
CLIM	Common Lisp Interface Manager
CMARPS	Combined Mating and Ranging Planning System
CNP	Contract-Net Protocol
CORBA	Common Object Request Broker Architecture
CSTAR	C-17 STEP Transfer and Retrieval
DARPA	Defense Advanced Research Project Agency
DART	Dynamic Analysis and Replanning Tool
DCOM	Distributed Component Object Model
DMSO	Defense Modeling and Simulation Office
DRMS	Defense Reutilization and Marketing Service
GDSS	Global Decision Support System
HLA	High Level Architecture
ICL	Interagent Communication Language
IIOP	Internet Inter-ORB Protocol
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
LOCIS	Logistics Command and Control Information Support
MASS	MICAP Asset Sourcing System
MICAP	Mission Capable Awaiting Parts
OAA	Open Agent Architecture
OMAR	Operator Model Architecture
OMG	Object Management Group
PDA	Personal Digital Assistant
RMI	Java Remote Method Invocation
RTI	Real-time Infrastructure
SAVE	Simulation Assessment Validation Environment
SCORE	Simulation Core Language
SFL	Simple Frame Language
SIMNET	Simulator Networking
SOM	Simulation Object Model
STEP	Standard for Exchange of Product Model Data

7. References

Agha, G. A. (1986). *ACTORS: A model of concurrent computation in distributed systems.* Cambridge, MA: MIT Press.

Bird, S. D. (1993). Towards a taxonomy of multi-agent systems. *International Journal of Man-Machine Studies 36,* 689-704.

Chavez, A., Dreilinger, D., Guttman, R., and Maes, P. (1997). A real life experiment in creating an agent marketplace. *Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology.* London, UK.

Cohen, P. R., Cheyer, A. J., & Wang, M. (1994). An open agent architecture. *1994 AAAI Spring Symposium.*

Davis, R. & Smith, R. (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence, 42,* 63-109.

Deutsch, S. E. (1993). Notes taken on the quest for modeling skilled human behavior. *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation.* Orlando, FL.

Deutsch, S. E., & Adams, M. J. (1995). The operator-model architecture and its psychological framework. *6th IFAC Symposium on Man-Machine Systems.* MIT, Cambridge, MA.

Deutsch, S. E., Adams, M. J., Abrett, G. A., Cramer, N. L. & Feehrer, C. E. (1993). *Research, Development, Training, and Evaluation (RDT&E) Support: Operator Model Architecture (OMAR) Software Functional Specification* (AL/HR-TP-1993-0027). Wright-Patterson AFB, OH: Armstrong Laboratory, Logistics Research Division.

Durfee, E. H. (1991). The distributed artificial intelligence melting pot. *IEEE Transactions on Systems, Man, and Cybernetics, 21,* 1301-1306.

Franklin, S. & Graesser, A. (1996). Is it an agent, or just a program? A taxonomy of autonomous agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages.* Springer-Verlag.

Freeman, B. (1997). *OMAR User/Programmer Manual, Version 2.0.* BBN Report No. 8181. Cambridge, MA: BBN Corporation.

Genesereth, M. R. (1992). An agent-based approach to software interoperability. *Proceedings of the DARPA Software Technology Conference.*

Genesereth, M. R. & Ketchpel, S. P. (1994). Software agents. *Communications of the ACM, 37,* 48-53.

Ginsberg, M. L. (1991). Knowledge interchange format: The KIF of death. *AI Magazine, 12, 3,* 57-63.

Graham, I. (1993). *Object Oriented Methods,* Second Edition. Menlo Park, CA: Addison-Wesley.

Gray, R. (1996). Agent Tcl: A transportable agent system. *Proceedings of the Fourth Annual Tcl/Tk Workshop,* Monterey, CA.

Green, S., Hurst, L., Nangle, B, Cunningham, P., Somers, F., Evans, R. (1997). *Software agents: A review.* Intelligent Agents Group Report, Department of Computer Science, Trinity College, Dublin.

Hewitt, C. & Inman, J. (1991). DAI betwixt and between: From "intelligent agents" to open systems science. *IEEE Transactions on Systems, Man, and Cybernetics, 21,* 1409-1419.

Jennings, N. & Wooldridge, M. (1996). Software agents. *IEE Review,* January 1996, 17-20.

Lesser, V. R. (1991). A retrospective view of {FA/C} distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics, 21,* 1347-1362.

Moran, D. B., Cheyer, A. J., Julia, L. E., Martin, D. L., & Park, S. (1997). Multimodal user interfaces in the open agent architecture. *Proceedings of the 1997 International Conference on Intelligent User Interfaces.*

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., & Swartout, W. (1991). Enabling Technologies for knowledge sharing. *AI Magazine, 12,* 3, 6-56.

Orfali, R., Harkey, D. & Edwards, J. (1996). *The essential distributed objects survival guide.* New York, NY: John Wiley & Sons.

Pan, J. Y. C. & Tenenbaum, J. M. (1991). An intelligent agent framework for enterprise integration. *IEEE Transactions on Systems, Man, and Cybernetics, 21,* 1391-1408.

Smith, R. & Davis, R. (1981). Frameworks for distributed cooperative problem solving. *IEEE Transactions on Systems, Man and Cybernetics, SMC-11,* 61-69.

Tsvetovatyy, M., Gini, M., Mobasher, B., & Wieckowski Z. (1997). MAGMA: Agent-based virtual market for electronic commerce. To appear in *Applied Artificial Intelligence, special issue on Intelligent Agents*, 1997.

Wooldridge, M. & Jennings, N. (1995). Agent theories, architectures, and languages: A survey. In M. Wooldridge & N. Jennings (Eds.), *Intelligent agents* (pp. 1-22). Berlin: Springer-Verlag.

Zlotkin, G. & Rosenschein, J. S. (1991). Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transactions on Systems, Man, and Cybernetics, 21,* 1317-1324.